



MÄLARDALEN UNIVERSITY

Department of computer science and electronic

BLIND SENSOR FOR SOCCER PLAYING ROBOTS

Master thesis work
15 credits, C-level

Author

DAVID CALVO
MIREN ITSASO MARTINEZ

Supervisor and examiner: **Lars Asplund**
Date: **7th June 2007**

Abstract

The present work, performed at Mälardalen University, illustrates the design of an electronic circuit to be implemented in a soccer playing robot. The main objective of this work is to make the robot know whether it is above a white line in the soccer field or not. This knowledge can help the robot to calculate the current position which it occupies.

Several infrared sensors are located in the bottom platform of the Robot in three different cards in order to detect the white lines. The three cards are part of a RS-485 network within the PPC which is supposed to receive the sensors' state from the cards. As the cards have to send the sensors' states to the PPC digitally while they are read in an analogue way, it is necessary to calculate each sensor's threshold which may change according to the field the robot is playing on.

The report, at first, considers relevant literature providing basic background knowledge on related works.

Afterwards, a detail analysis of the problem and the method used are presented. Moreover, the algorithm of the program is examined along with the final code and the way this has been tested. Schematic and layout are also included.

Finally, an analysis of the solution obtained is exposed. Although this thesis does not explain how the specific network works, it illustrates how the designed code works in a simulation kit. An analysis of the results obtained and the conclusions are presented along with the suggestions on the improvements that could be made in the design.

Preface

Express our thankfulness to Lars Asplund for giving us the opportunity to learn and being so patient.

Table of contents

ABSTRACT	2
PREFACE	3
TABLE OF CONTENTS	4
1. INTRODUCTION	5
1.1. PURPOSE	5
1.2. BACKGROUND	6
2. RELEVANT THEORY	6
2.1. INFRARED SENSOR	6
2.2. ATMEGA16	8
2.2.1. Overview	8
2.2.2. USART	9
2.2.3. ADC	11
2.3. C PROGRAMMING KNOWLEDGE	12
2.4. RS485	13
2.5. KNOWLEDGE OF THE REQUIRED SOFTWARE AND HARDWARE	15
2.6. PREVIOUS DESIGN	16
3. THESIS FORMULATION	17
4. DESIGN	18
4.1. ANALYSIS OF PROBLEM	18
4.2. HARDWARE, SCHEMATIC AND LAYOUT	21
4.2.1. Required components for one card	21
4.2.2. Schematic and layout	21
4.3.2. Code generation	24
4.3.2.1. Flux diagram & Algorithm	24
4.3.3. Test	28
4.2. MODEL/METHOD	29
4.3. SOLUTION	30
5. RESULTS, ANALYSIS OF RESULTS, RECOMMENDATIONS, FUTURE WORK	30
6. SUMMARY AND CONCLUSIONS	31
7. REFERENCES	33
8. APPENDIX	35
APPENDIX A. C CODE	35
APPENDIX B. SCHEMATIC	38
APPENDIX C. LAYOUT	39
APPENDIX D. PREVIOUS SCHEMATIC	40
APPENDIX E. REQUIRED SOFTWARE AND HARDWARE	41

1. Introduction

1.1. Purpose

In MDH robot laboratory, there are five robots constituting a football team taking part in the international robot soccer competition in Atlanta in July 2007. This report considers a small part of a single robot.

In the bottom platform of the soccer playing robot, there are 24 infrared sensors. When the sensors are above a white line, they give a signal due to the infrared signal will be reflected. On the other hand, weaker signals will be received when the sensors are above a green colour field. A general purpose of this thesis work is shown in figure 1-1. As the cards have to send the sensors' states to the PPC digitally while they are read in an analogue way, it is necessary to calculate each sensor's threshold which may change according to the field the robot is playing on.

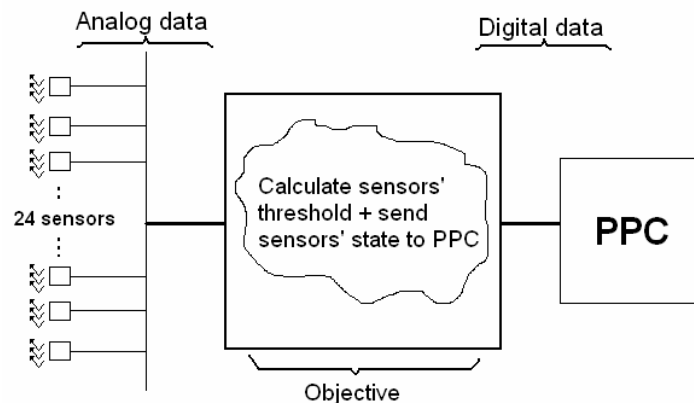


Figure 1-1. Description of the main objective

In the aforementioned competition, the robot is going to play on different kinds of soccer fields. Depending on the field, the sensors threshold may change, thus the system must automatically calculate each sensors' threshold before taking any course of action.

One of the requirements that the thesis has to comply with, is that the sensors must be dispersed in three different cards in order to lie over the entire platform as shown in figure 1-2. Thus, it is necessary to establish a route of communication between the three cards and the PPC, which is going to receive all the information from the cards.

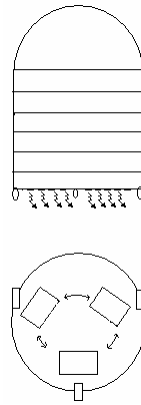


Figure 1-2. 24 sensors divided in 3 cards

In brief, the purpose of this report is to illustrate how the work has been developed, explaining the programmed code's algorithm, mentioning the schematic and the tests as well.

1.2. Background

The need of implementing this project derives from the fact that a model with similar aims had been designed however not offering enough benefit. The main problem encountered was that when there was a change of playing field, it was necessary for the cards to be calibrated manually using a potentiometer. In the following section there will be a brief explanation of how the old design works.

2. Relevant theory

2.1. Infrared sensor

OPB607 sensors from OPTEK are the infrared sensors chosen for this application. It consists of an infrared emitting diode and a NPN silicon photo Darlington mounted "side by side" on parallel axes in a gray opaque plastic housing. Both the emitting diode and photo Darlington are encapsulated in a filtering epoxy to reduce ambient light noise. The photo Darlington responds to radiation from the emitter only when a reflective object passes within its field of view. [1]

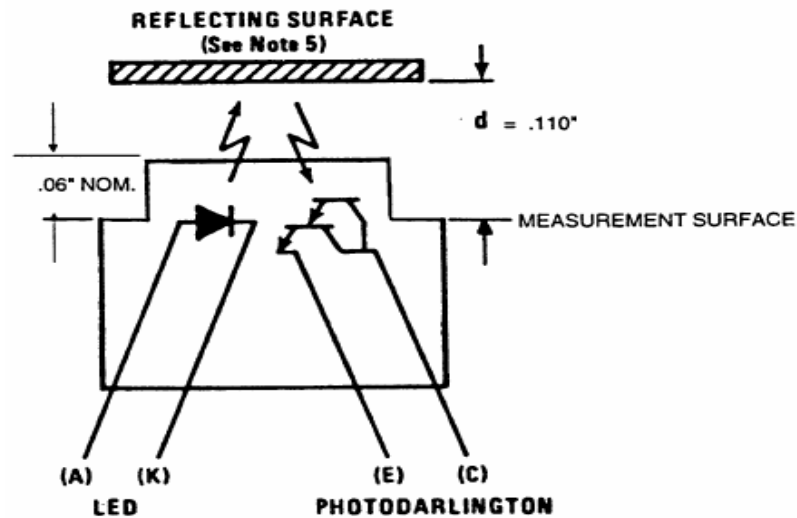


Figure 2-1. Above a reflection surface

In this application, the reflecting surface is the white line in the soccer field. Both the diode and the photodarlington have 75mW power dissipation.

It is important to take into consideration the distance from the assembly measurement to the reflective surface. In figure 2-2 is appreciated how the sensor's normalized functionality move away as much as the distance is longer.

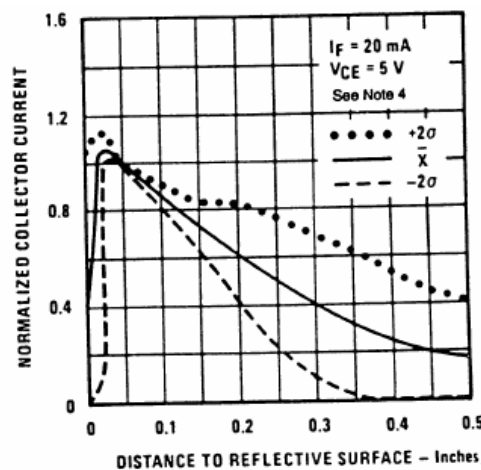


Figure 2-2. Normalized collector Current vs. Objective

The collector current and consequently the emitter current will depend on the diode forward current as it shows figure 2-3. Taking into consideration the application it is working with, although not having changes in the diode forward current, any change in the reflecting surface, it will determinate also the emitter current, so the value that the emitter current will have it is going to depend on the field where the robot is playing.

That means that the signal received from the sensors might be different in each football field.

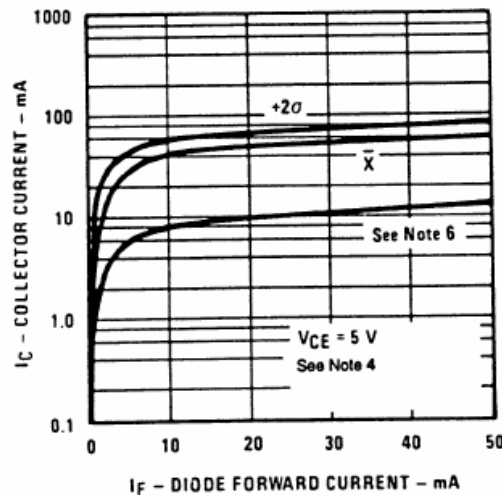


Figure 2-3. Collector current vs. Diode Forward Current

2.2. Atmega16

Necessary information concerned to Atmega16 microcontroller can be found in the microcontroller's data sheet. Below, it is exposed the explanation about the most important features used in the thesis. [2]

2.2.1. Overview

ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture.

Below is a list with the features used that Atmega16 provides:

- 16K bytes On-chip In-System Reprogrammable Flash memory for program storage.
- 512 bytes EEPROM
- 1K byte SRAM,
- 32 general purpose I/O lines and 32 general purpose working registers
- JTAG Interface and On-chip Debug
- A serial programmable USART
- An 8-channel, 10-bit ADC
- A programmable Watchdog Timer with Internal Oscillator.

Among these feature, only the more significant details about USART and ADC are going to be exposed.

2.2.2. USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device.

Figure 2-4 illustrate USART's communication frame format.

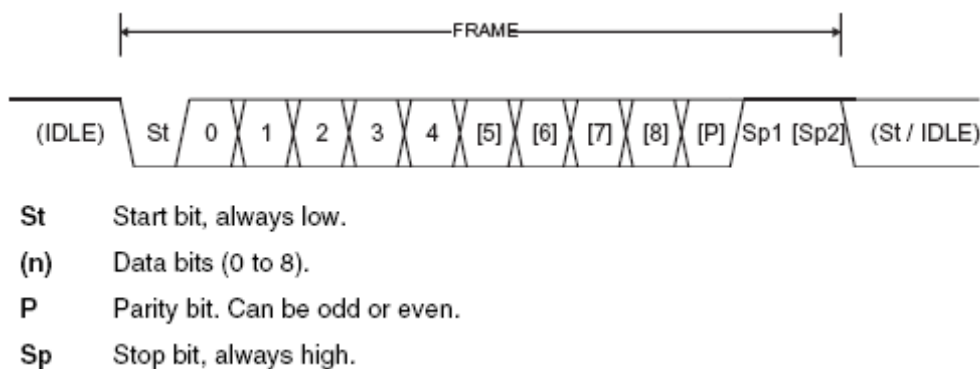


Figure 2-4. USART frame

The usual form of serial communication, and concretely the one that is discussed, is the asynchronous serial communication. It is asynchronous in the senses that a common clock signal is not required at both the transmitter and the receiver in order to synchronize the data detection. [6]

A frame starts with the start bit followed by the least significant data bit. Then the next data bits, up to a total of nine, are succeeding, ending with the most significant bit. If enabled, the parity bit is inserted after the data bits, before the stop bits. When a complete frame is transmitted, it can be directly followed by a new frame, or the communication line can be set to an idle (high) state. Figure 2-4 illustrates the possible combinations of the frame formats. Bits inside brackets are optional.

The frame format used by the USART is set by the UCSZ2:0, UPM1:0, and USBS bits in UCSRB and UCSRC. The Receiver and Transmitter use the same setting. Note that changing the setting of any of these bits will corrupt all ongoing communication for both the Receiver and Transmitter.

The USART Character SiZe (UCSZ2:0) bits select the number of data bits in the frame. Moreover the USART Parity mode (UPM1:0) bits enable and set the type of parity bit. The selection between one or two stop bits is done by the USART Stop Bit Select (USB) bit. The receiver ignores the second stop bit.

In addition, setting TXEN, in the UCSRB register, transmission is enabled. Before a transmission it should do nothing until UDR is ready for more data to be written to it. This can be checked with the UDRE bit in UCSRA register. The transmit buffer can only be written when the UDRE Flag in the UCSRA register is set. Any data written to UDR when the UDRE Flag is not set will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

On the other hand, setting RXEN, in the UCSRB register, receiving is enabled. The USART Receiver has one flag that indicates the receiver state. Therefore the Receive Complete (RXC) Flag indicates if there are unread data present in the receive buffer. This flag is one when unread data exist in the receive buffer, and zero when the receive buffer is empty (i.e., does not contain any unread data). If the receiver is disabled (RXEN = 0), the receive buffer will be flushed and consequently the RXC bit will become zero.

Receive and transmit buffer can be read and write by the UDR I/O location.

The UBRRH register shares the same I/O location as the UCSRC register. Therefore some special consideration must be taken when accessing this I/O location. When doing a write access of this I/O location, the high bit of the value written, the USART Register Select (URSEL) bit controls which one of the two registers that will be written. If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated.

Finally, register UBRR must also be set. The formula in figure 2-5 calculates the value for this register.

Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

Figure 2-5. Calculate the UBRR register value

Thus, the only issue confronting the programmer is to ensure that the serial communication parameters of both the transmitter and the receiver match. Specially, this includes setting the correct number of data bits (usually 8), determining whether or not to include a parity bit (usually not), and setting the baud rate. [6]

2.2.3. ADC

The ADC is connected to an 8-channel Analogue Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The ADC converts an analogue input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. It generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted.

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. This must be set in the ADCSRA register.

There exists more than one mode to do the conversion such as, auto triggered mode, free running mode or single conversion. Latter is used and explained in the following lines.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry as shows the figure 2-7. In addition, in the figure 2-7 can see that the actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of a first conversion. That why it is recommendable to ignore the first conversion (figure 2-6).

When a conversion is complete, the result is written to the ADC Data Registers. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge.

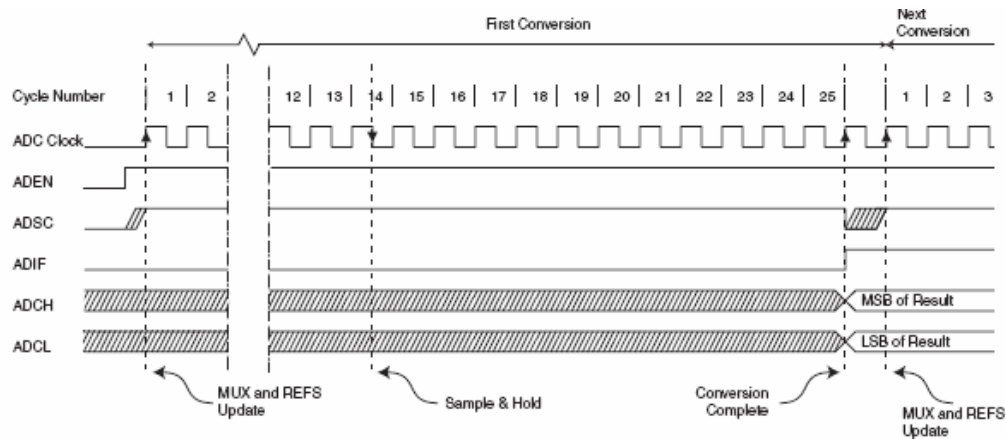


Figure 2-6. ADC Timing Diagram, First Conversion (Single Conversion Mode)

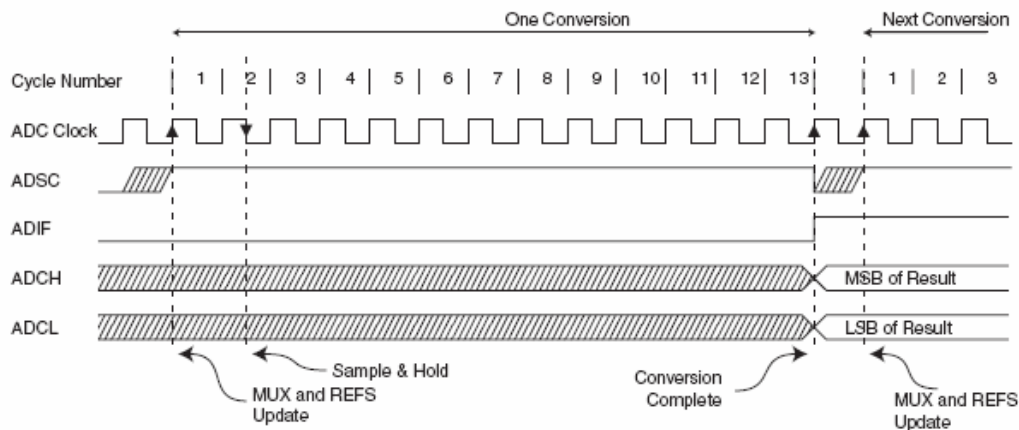


Figure 2-7. ADC Timing Diagram, Single Conversion

The selection of the channel to be converted must be done by the MUXn and REFS1:0 bits in the ADMUX Register. The channel has to be selected before start the conversion.

2.3. C programming knowledge

There are some links related with C programming in the reference section.
[6][10][11]

2.4. RS485

The main information sources about the RS-485 network are mentioned in the reference. [8] [15]

EIA-485 (formerly RS-485 or RS485) is an OSI Model physical layer electrical specification of a two-wire, half-duplex, multipoint serial connection. The standard specifies a differential form of signaling. The difference between the wires' voltages is what conveys the data. One polarity of voltage indicates logic 1 level; the reverse polarity indicates logic 0. The difference of potential must be at least 0.2 volts for valid operation, but any applied voltages between +12 V and -7 volts will allow correct operation of the receiver.

EIA-485 only specifies electrical characteristics of the driver and the receiver. It does not specify or recommend any data protocol. EIA-485 enables the configuration of inexpensive local networks and multidrop communications links. It offers high data transmission speeds (35 Mbit/s up to 10 m and 100 kbit/s at 1200 m). Since it uses a differential balanced line over twisted pair (like EIA-422), it can span relatively large distances (up to 4000 feet or just over 1200 meters).

EIA-485 drivers need to be put in transmit mode explicitly by asserting a signal to the driver. This allows EIA-485 to implement linear topologies using only two lines and between the ribbon cable. The purpose of this signal is to connect the driver to its output terminals, A and B (Figure 2-8). If the "Enable" signal is OFF, one can consider the driver as disconnected from the transmission line. An RS-485 driver must have the "Enable" control signal when for instance in RS-422 driver is not always required.

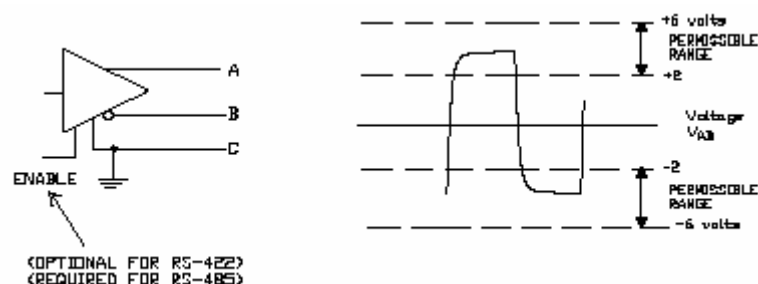


Figure 2-8. Balanced Differential Output Line Driver

The recommended arrangement of the wires is as a connected series of point-to-point (multidropped) nodes, a line or bus, not a star, ring, or multiply-connected network. Ideally, the two ends of the cable will have a termination resistor connected across the two wires. Without termination resistors, reflections of fast driver edges can cause multiple data edges that can cause data corruption. Termination resistors also

reduce electrical noise sensitivity. The value of each termination resistor should be equal to the cable impedance (typically, 120 ohms for twisted pairs). Star and ring topologies are not recommended because of signal reflections or excessively low or high termination impedance.

Figure 2-9 shows the potentials of the '+' and '-' pins of an RS-485 line during transmission of an RS-485 byte:

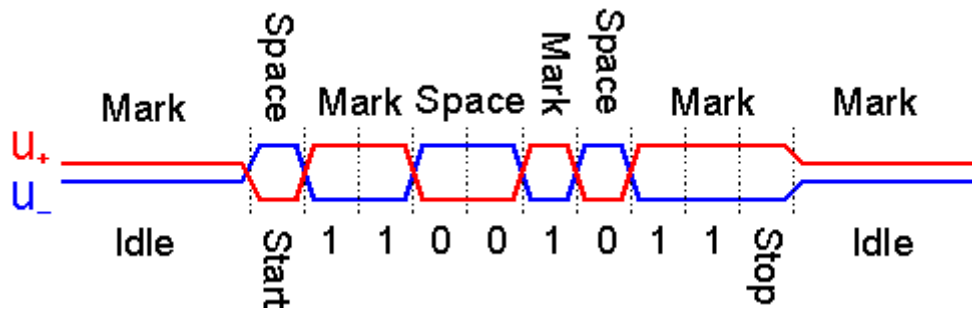


Figure 2-9. RS-485 byte transmission

The table below illustrates a summary of the RS485 characteristic.

RS485	
Differential	yes
Max number of drivers	32
Max number of receivers	32
Modes of operation	half duplex
Network topology	multipoint
Max distance (acc. standard)	1200 m
Max speed at 12 m	35 Mbs
Max speed at 1200 m	100 kbs
Max slew rate	n/a
Receiver input resistance	$\geq 12 \text{ k}\Omega$
Driver load impedance	54 Ω
Receiver input sensitivity	$\pm 200 \text{ mV}$
Receiver input range	-7..12 V
Max driver output voltage	-7..12 V
Min driver output voltage (with load)	$\pm 1.5 \text{ V}$

Figure 2-10. RS-485 network characteristic

The implementation on the software side is not much more difficult than with RS232. It is the reason why RS485 is so popular with computers, PLCs, micro controllers and intelligent sensors in scientific and technical applications.

Figure 2-11 illustrates the multiple-transceivers RS-485 which is going to be implemented. It is important to keep the distances from the twisted pair to the receivers as short as possible.

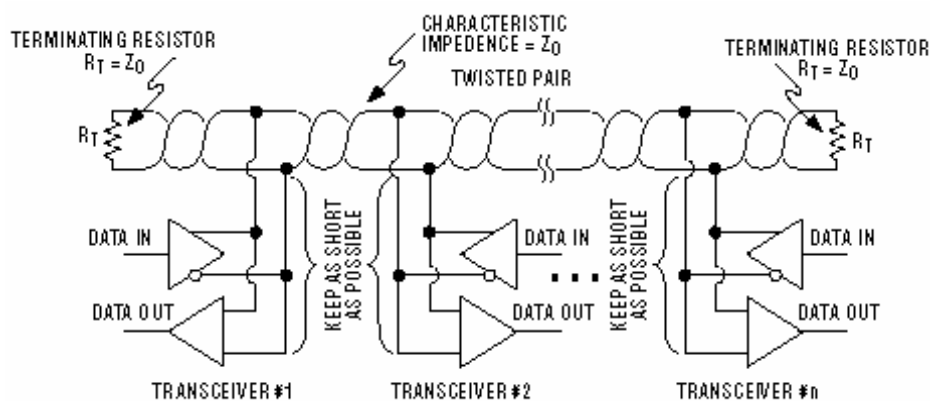


Figure 2-11. Multiple-transceiver RS-485 network

2.5. Knowledge of the required software and hardware

As the knowledge on the software as well as the several devices used were null and never used before, it has been necessary to take advantage of either the help of the corresponding software or manuals found in internet mentioned in the reference section. Appendix E illustrates all the information about the necessary software and devices.

2.6. Previous design

The old system tests the field and it also gives back whether there is a white line under the robot or not. The system is composed by three equal cards. The schematic corresponding to a single card is exposed in appendix D.

When there is a change of playing field, it is necessary for the cards to be calibrated manually using a potentiometer.

Since there are eight sensors that they will have a single manual calibration system, it may be impossible to set the optimal value for all sensors.

The sensors tolerance will depend on the signal called *level*. In addition, point 1 of the schematic (figure 2-12) illustrates how this signal is set. Making this calibration will take some time and it doesn't guarantee any precision because human error will be implied among other things.

The mentioned manual calibration system is designed with a variable resistance, a transistor and a passive filter so far. Using the variable resistance the transistor base current can be modified. At the same time this current fixes the collector and emitter current. This emitter current is transformed into a voltage by the filter stabilizing the *level* signal.

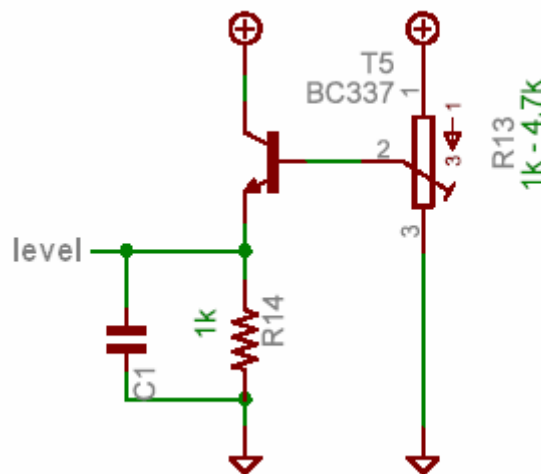


Figure 2-12. Point 1 in the schematic

The transistor in the second point of the schematic is working like a switch whose state depends on the gate voltage that is set by the signal called *load*. Therefore when the switch is ON, the *level* signal flows from the drain to the source pin generating a current in the bases of the transistors located in point 3 of the schematic. Consequently, the current is going through the sensors' infrared emitting diodes taking eight samples, one for each sensor.

Finally, in point 5, the shift register swaps data between cards and the Power PC. The mentioned shift register takes data from the pins A to H and it sends this data (serial) from the pin QH to the socket arriving until the PPC.

3. Thesis formulation

Before advancing in the implementation phases of the thesis project, it is necessary to acquire knowledge on related work. Once this part is completed, the main task is to write the code to be implemented in the microcontrollers, design the circuit hardware, test the code, and if there is any time left, built the boards.

The work done within the 10 p thesis' work was divided into 5 different main parts:

- 1- Gathering information and acquiring all the necessary knowledge: This part involves basically the consideration of everything mentioned in section 2.
- 2- Suggesting an idea of how the program and the hardware will be structured and the designation of the schematics and the layout.
- 3- Creating and compiling the C code.
- 4- Testing the code and returning to step 3 to rewrite it until the application works as it is supposed to and building the boards if there is any time left.
- 5- Reporting and presenting the outcome of the project.

4. Design

4.1. Analysis of problem

In figure 4-1, we have a general circuit. The three cards as well as the PPC are connected to the RS485 network. The three cards are identical to each other so each one has eight sensors and an Atmega16 microcontroller. The Max 489 is a transceiver; it connects the microcontrollers to the RS-485 network.

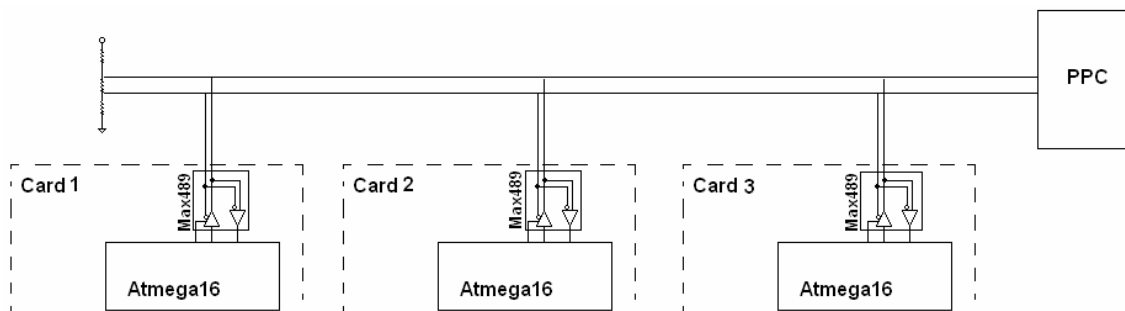


Figure 4-1. General diagram

The microcontroller can do 2 basic things:

- 1—Calibrating: this means calculating each sensor's threshold.
- 2—Sends the sensor's state digitally to the PPC, i.e. in the form of a logical 1 or 0.

Any action taken by the microcontroller is a result of a PPC's order. The PPC is sending messages to the RS485 network. The structure of the data frame is illustrated in figure 4-2:

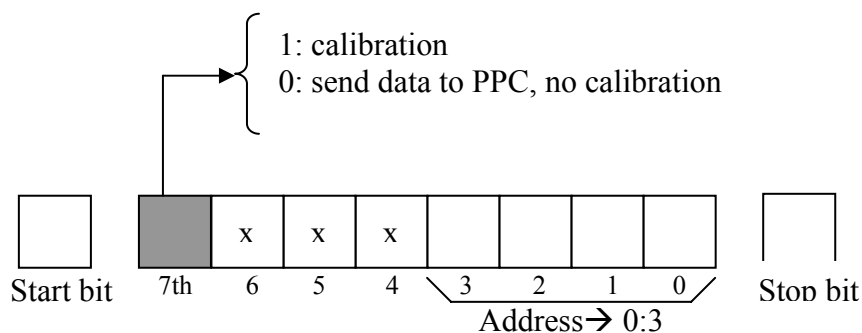


Figure 4-2. Order's location in the frame

Therefore the frame consists of a start bit, followed by the eight bit data and a 1 stop bit. The parity bit is not used and the communication is 9600bps with an external ceramic oscillator of 16MHz.

It is important to avoid any kind of collision risk because of the fact that only one device can be speaking in the network. Thus, the microcontrollers speak only when the PPC orders them. Therefore, the first thing the microcontroller has to do is to check if the frame it has received contains its address in the 4 less significant bits.

The next step that the microcontroller has to check is what it is being ordered to do; a calibration or transmission of the sensors' state. This is specified, checking the most significant bit.

Let's suppose that the robot is reset and it has to start playing in the same field that it did last time. In this case, it is not necessary to calculate the sensor threshold again because it has already been calculated. In order not to lose this information, when a calibration is done the microcontroller is writing this values in the EEPROM memory. This way, after a reset and before starting playing, the robot will read the calibration value stored in the EEPROM without the necessity of making the calibration again.

The sensors' values are going to be read from the microcontrollers' analogue input. However, the PPC wants to receive the data digitally as figure 4-3 shows.

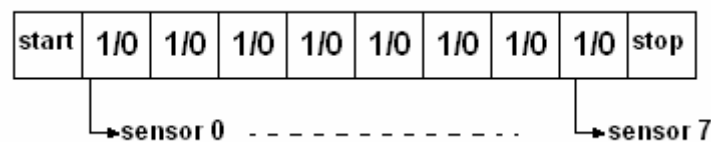


Figure 4-3. Each sensor's state in the frame

The bit value will be a logical 1 or 0 if the value read from the ADC is bigger or smaller than the corresponding sensor's threshold.

Figure 4-4 shows a general flux diagram representing the concepts mentioned above. Below the diagram, the algorithm of the program is taken into consideration.

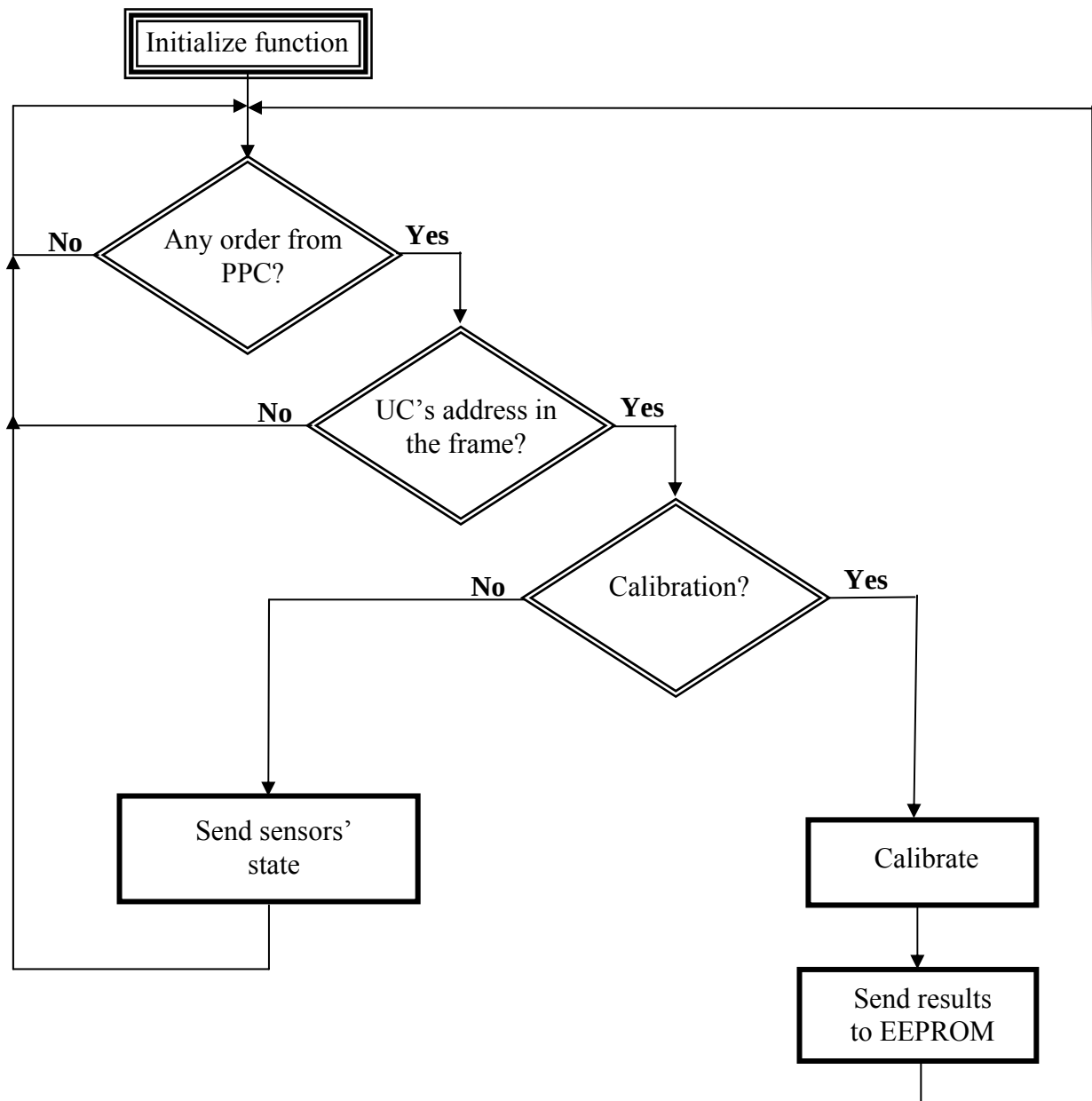


Figure 4-4. General flux diagramme

Thus, the application requires at least six functions to perform the following tasks:

- A initialization function
- A function for making the calibration
- A function for sending data to EEPROM
- A function for sending the sensors' states
- A function for reading data from sensors
- A main function

4.2. Hardware, schematic and layout

4.2.1. Required components for one card

- ⇒ Atmega16 microcontroller
- ⇒ Max489 transceiver
- ⇒ 8 OPB607 infrared sensors
- ⇒ 2 sockets RS485
- ⇒ JTAG pin out
- ⇒ BC337 NPN transistor
- ⇒ Ceramic oscillator, 16MHz
- ⇒ 5 capacitors, 0.1 μ F
- ⇒ 14 resistors
 - 1 resistor, 2.2k Ω
 - 4 resistors, 120 Ω
 - 9 resistors, 10k Ω
- ⇒ Cable RS485

4.2.2. Schematic and layout

The emitters of the infrared sensors' transistor are connected to the microcontroller's ADC input. At the same time the cathodes of the sensors' diode is connected to the collector of a NPN transistor's pin. The purpose of this is to consume exactly the necessary current from the sensors. The base of the transistor is connected to the PD3 pin in the Atmega16, thus, that pin can be set just before reading the sensors' values and be disconnected when reading is finished, saving the unnecessary current and the 75mW that both the diode and the photo Darlington dissipated.

The reference voltage for the ADC (V_{REF}) in the Atmega16 indicates the conversion range for the ADC. In this case, it is connected to Avcc, which is 5 Volts.

There are some reasons for the fact that 4 capacitors are connected in parallel. Firstly, in order for the interferences not to enter the CI in question, since having a digital device such a microcontroller can alter its normal functionality by changing the value of some registers. These capacitors also avoid the bad effects that the commutation noise produces. As the Atmega16 is requesting load of several mA in its pins, it has to raise the voltage of all or several pins' to the tension of the feeding demanding a major consumption than needed. At the time of the connection, a weakening of the tension will take place in the feeding of the microcontroller, producing a not as fast as expected response in the output of the pins, and for this cause and for the purpose of not creating current loops, the capacitor must always be as close to the pins as possible.

On the other hand, the AVCC pin is the supply voltage pin for Port A and the A/D Converter, and it should be externally connected to V_{CC} . [1]

The MAX489 transceiver's pins RO, DI, and DE are connected to the microcontroller's pin PD0, PD1 and PD2 respectively.

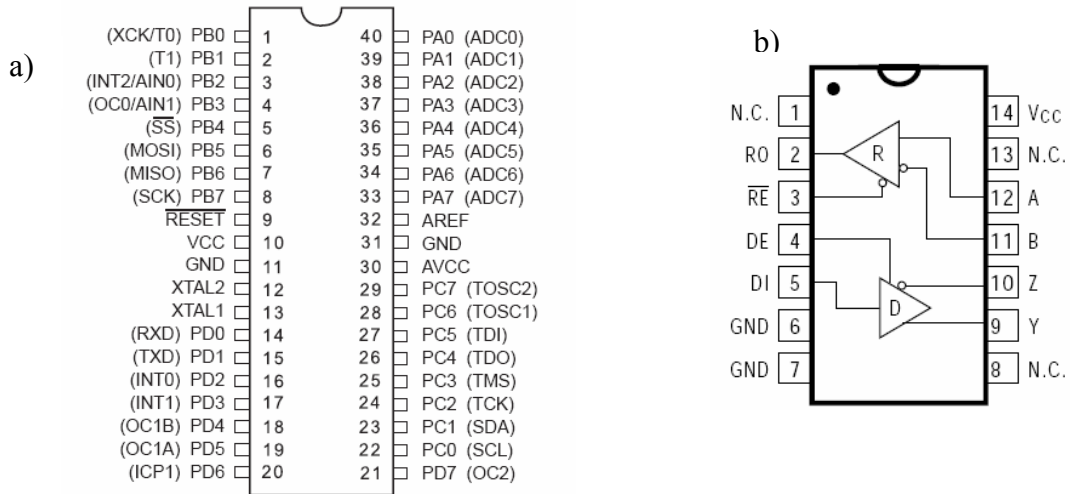


Figure 4-5. Atmega16 and Max489 pin configuration

Figure 4-5 is the pin configuration of Atmega16 and MAX489 transceiver. The PD0 pin is used as an RXD pin to receive data using USART (The Universal Synchronous and Asynchronous serial Receiver and Transmitter). In the same way the PD1 pin, also called TXD, is going to be the transmitter pin. The PD2 pin is configured as an input and is connected to the transmission *Enable* pin of the transceiver. The purpose of using this *Enable* signal is to connect the driver to its output terminals, A and Y. If the *Enable* signal is OFF, one can consider the driver as disconnected from the transmission line. An RS-485 driver must have the *Enable* control signal. [3]

Finally, the connection between the transceiver and the RS485 network is made using the sockets shown in figure 4-6.

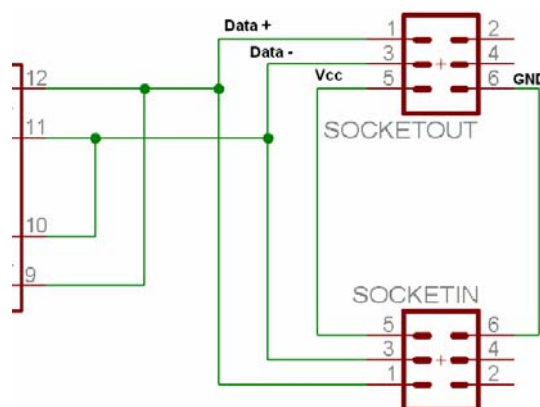


Figure 4-6. Connection between MAX489 and sockets

On the other hand, the figure 4-7 explains how to connect the JTAG pins to the Atmega16. The JTAG is used to program the microcontroller.

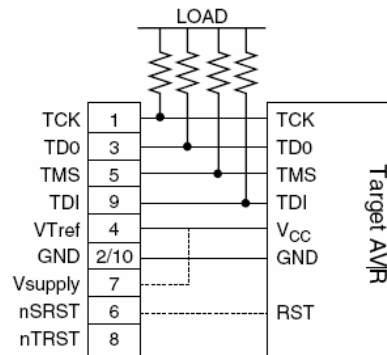


Figure 4-7. Connecting JTAG ICE to Target Board

Appendix B shows the layout of the circuit made in Eagle. The shape of the card was chosen to be that one in order that it can change neither the sensor locations in the platform of the old design nor the size that the cards hold in the robot bottom platform. In figure 4-8, the sensors' positions are indicated by circles.

The board has just two layers and the wire's width is 16mil.

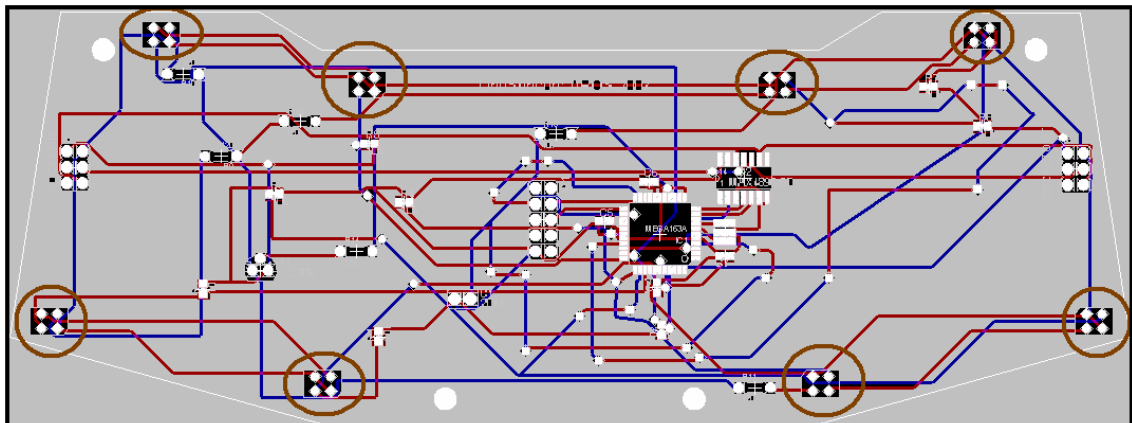


Figure 4-8. Sensors' position in the layout

4.3.2. Code generation

4.3.2.1. Flux diagram & Algorithm

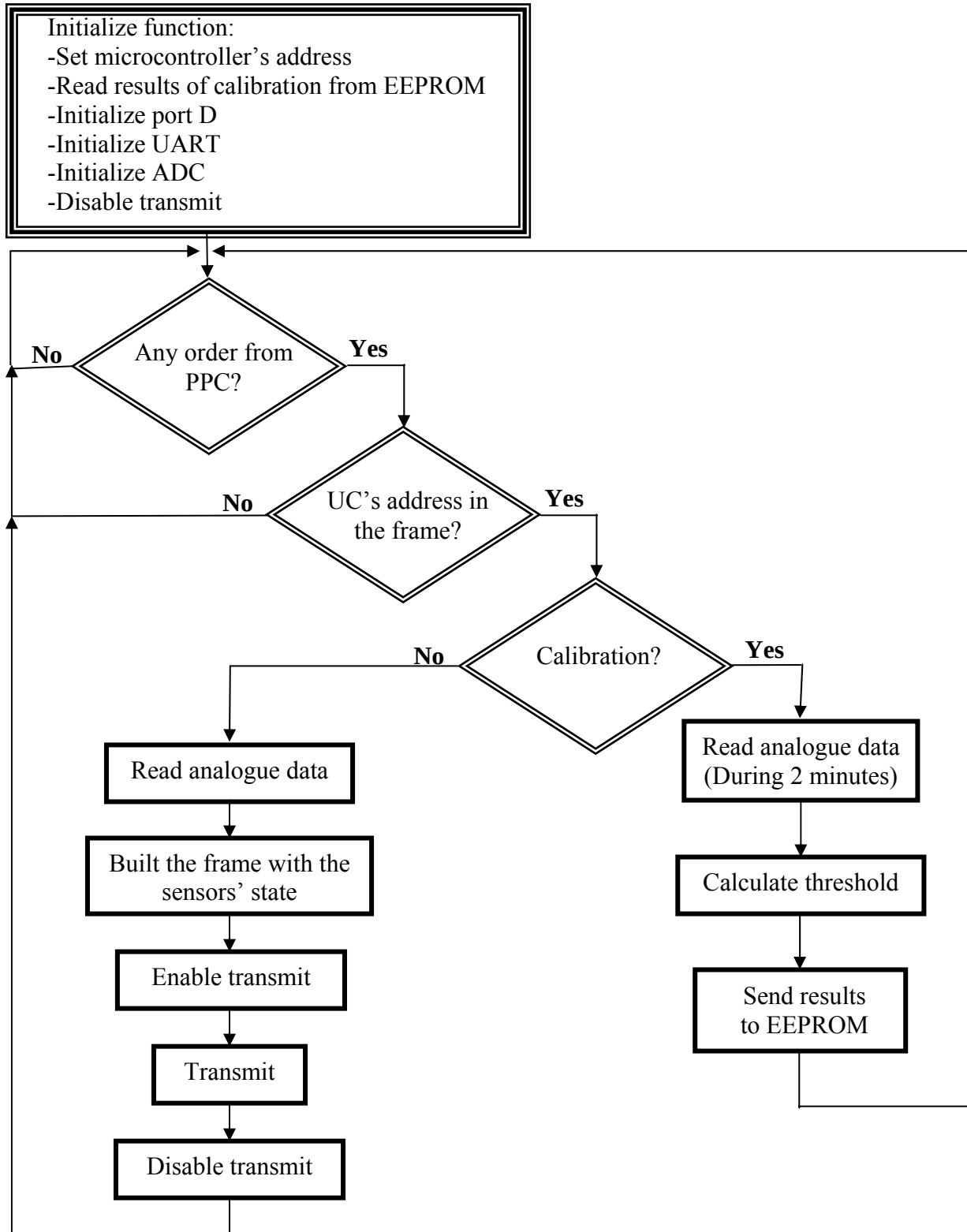


Figure 4-9. Programme flux diagram

In this section the algorithm of the program is going to be explained. The resultant C code can be found in Appendix A.

Main function

```
Call to init_function
While (forever)
    Reset watchdog
    Do nothing until data have been received and is ready to be read from UDR
    Read UDR
    If (address=uc's address or broadcast) do:
        If (calibration) do:
            Call calibrate function
            Call to the function that sends calibration results to EEPROM
        Else do:
            Call to the function that sends through USART the sensors' state
        End if
    End if
End while
```

Initialize function

```
Enable watchdog (2seg)
Set microcontroller address
For (8 steps) do:
    Read calibration results from EEPROM
    Reset watchdog
End for
Initialize PORTB. Pin 2 and 3 as output
Initialize USART. 16MHz and 9600 bps
Initialize ADC. Enable ADC and prescaler 125kHz
Ignore the first conversion of the ADC
Disable transmission
```

Calibration function

Before illustrating the algorithm, an explanation of how the threshold of a single sensor is calculated, is taking place.

First of all, as figure 4-10 shows, *xprima* value is calculated, being the half-sum of the biggest and the smallest value read during the first half time period of the calibration. However, this is not the optimal sensor's threshold, thus in the next half period of the calibration, a more concrete value is calculated taking advantage of knowing the value of *xprima*.

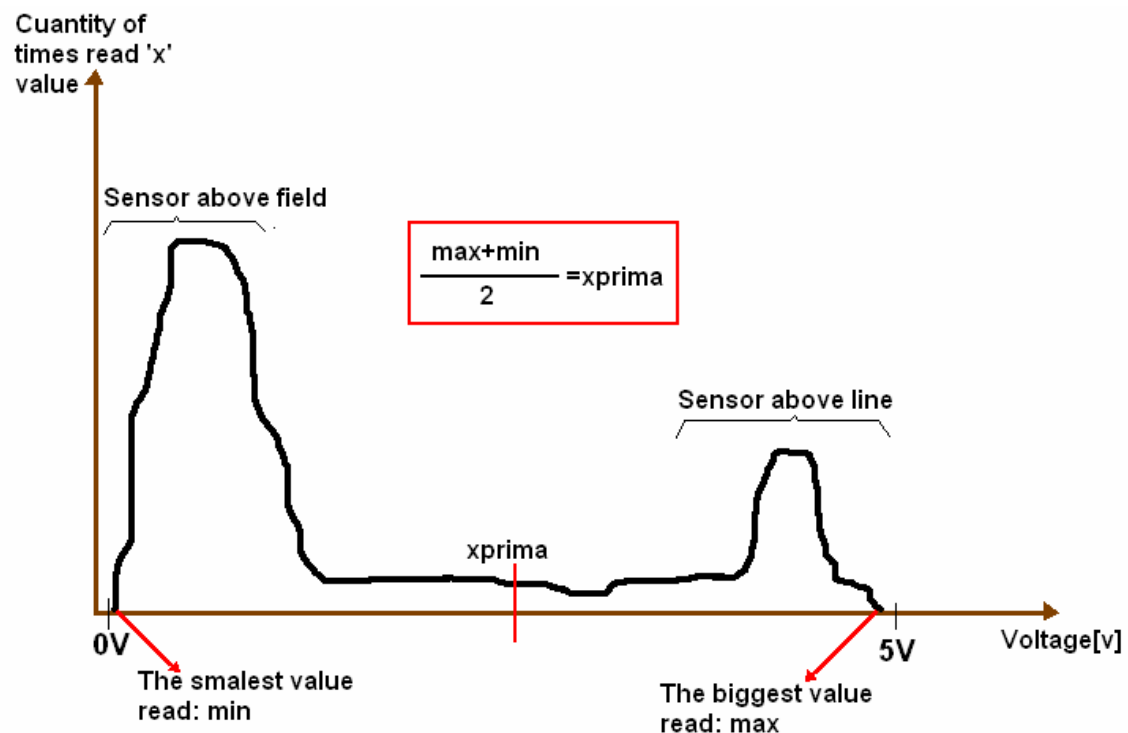


Figure 4-10. X prima calcule

Then, the Atmega16 calculates the average of all the values read when the sensor is above the field, i.e. the average of all the values bigger than *xprima*. The same calculation is taking place when the sensor is above a white line, i.e. the average of all the values that are smaller than *xprima*. After that, the two values obtained are half-summed obtaining a closer result to the optimal threshold value.

Below follows the algorithm that calculates the 8 sensors' threshold.

```

For (1000 steps)(1000 samples) do:
  Watchdog reset
  For (8 steps)(for 8 sensors) do:
    Read analog data
    If(data read bigger than max) do:
      Max=data
    End if
    If (data read smaller that min)do:
      Min=data
    End if
    Xprima=(max+min)/2
  End for
  Delay 50m sec.
End for
  
```

```
For(1000 steps)(1000 samples) do:
  Watchdog reset
  For (8 steps)(for 8 sensors) do:
    Read analog data
    If (read data bigger than xprima) do:
      Increment line counter
      Sum_line=Sum_line+read data
      Line values average= sum_line/line counter
    Else do:
      Increment field counter
      Sum_field=sum_field/field counter
      Field values average= sum_field/field counter
    End if
  End for
  Delay 50m sec.
End for
```

Read data function

```
Watchdog reset
Select the channel for read
Enable transistor
Delay 1m sec.
Start conversion
Do nothing until the conversion is finished
Disable transistor
Return value
```

Function that sends sensor' state

```
Initialize variable 'dades'
For (8 steps)
  Read data
  If (data read is bigger than the sensor's threshold) do:
    Write logic 1 in the corresponding bit of the 'dades' variable
  Else do:
    Write a logic 0 in the corresponding bit in the 'dades' variable
End for
Do nothing until UDR is ready for more data to be written to it
Enable transmit
Transmit 'dades'
Do nothing until the transmitting is finished
Delay 30u sec.
Disable transmit
```

4.3.3. Test

AVR Studio and Hyper terminal software within the evaluation kit STK 500 are used for the tests to be made.

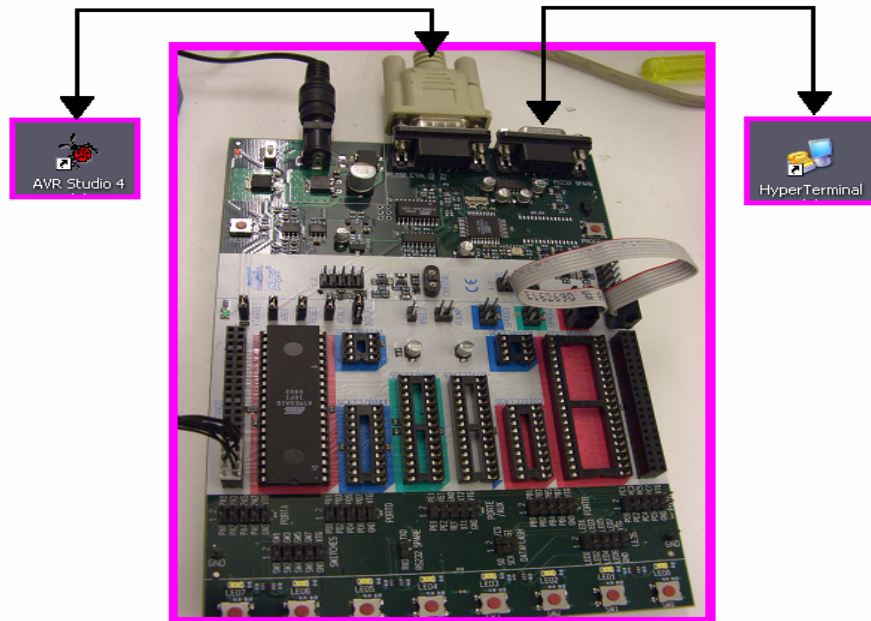


Figure 4-11. STK500 simulation kit

Some small differences between the real board and the board used to test the code are expected. First of all, the communication will be RS232 instead of RS485 as it is the one that the STK500 provides, and this way the computer will make the task in correspondence to the PPC. Instead of using an external ceramic oscillator of 16MHz, an internal RC oscillator of 4MHz is used. Therefore, it is important to set the corresponding fuses before programming the microcontroller in AVR Studio, and to change the UBRRL register in the code, from value 0x67 to 0x19 as well as the ADCSRA register from value 0x87 to 0x85.

The microcontroller is programmed with SPI from AVR Studio4. The code has been divided into different parts which having been analyzed separately. Once each part is working, all parts were attached and tested together.

The parts in which the program has been divided into are the following:

- UART, checking the communication.
- The ADC
- W/R EEPROM
- Calculating the frame to send to the PPC
- Calibrating

The ADC test has been done with some voltage divisor as shown in figure 4-12.

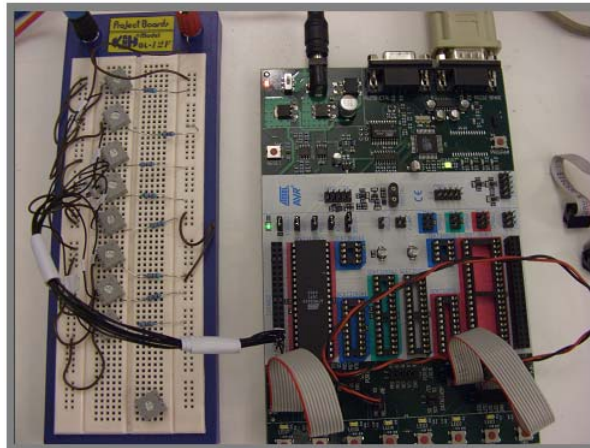


Figure 4-12. STK500 testing the code

For instance, we can test the code writing in the Hyper Terminal panel the character ' '; this ASCII character correspond to the 0x81 hexadecimal value, or the 10000001 binary one. Thus, taking the imposed frame protocol into consideration, we have that the microcontroller with address '0x01' must do the calibration. Then, 2 minutes (calibration time) later, the PC may send the ASCII character 'q' corresponding to the 01110001 binary value therefore, now, the Atmega16 has to reply to the PC sending the sensors' states.

Assume that the PC receives a 'C' character, which has a 0x43 hex value corresponding to the ASCII table. In this case, the sensors 0, 1 and 6 are above a white line in contrast to the rest of the sensors which lie above the field.

4.2. Model/method

As the knowledge on the Atmega16 as well as the software used was null and never used before, much time has been spent gathering information and looking into what is done and how can be done.

The next step is to generate the schematic and the layout of the board using Eagle software. As this project's intention was to improve the previous one, the need to calculate automatically the sensors' threshold urges us to use a microcontroller. The justification of using the Atmega16 from Atmel, the RS485 network and all the hardware aspects and specifications, is something beyond the purpose of this thesis; all the aforementioned data have been selected by the IDE department.

Once the first draft of the code was finished in WinAVR using C language, the code was compiled and built. AVR Studio software is used to make the code simulation. This software is now fully compatible with the latest version of Win AVR. The former 64K limit on flash size when using Win AVR/GCC compiler has been fixed. It requires

that one uses the latest Win AVR release (after 2007-01-02). However, the either analogue or UDR register simulation has not yet been supported, and that was not sufficient to ensure the correct operation of the code.

Urged by the need to communicate and send the sensors' state, Atmega16 offers the USART or the SPI serial communication. But normally the SPI bus is used for very short distance communication with peripherals or other microcontrollers that are located on the same circuit board. This is different from the UART, which is used for communication over longer distances, such as between units or between microcontroller and PC. The SPI was developed to provide relatively high-speed, short distance communication using a minimum number of microcontroller pins. That is why the USART was chosen. [6]

4.3. Solution

The schematic, the layout and the code can be found in the appendices. The time spent in this project was enough to make the program work as expected, thus the purpose of this thesis has been almost achieved. Nevertheless, much more things could have been achieved in a longer period of time, as the use of the microcontroller provides us with certain abilities.

It seems that all things mentioned above are working as they were expected. The report focuses on one single board, although the author's strongest aim was to make 3 cards and the PPC work in the same network. Given that a correct performance in the system consisted of the PC and the simulation kit has been verified by tests, the entire network is highly likely to work.

5. Results, analysis of results, recommendations, future work

As the STK500 responds right to the PC orders, it is verified that, at least, the code is correct.

The code mainly includes a while loop which, depending on what data it receives from USART, will call the respective function. It is also evident that the non-interrupts provided by Atmega16, are not used. The reason is that the microcontroller has no other task to perform; therefore there is nothing wrong in waiting an order. For this specific application, the interrupts are not potentially useful. However, both ADC and USART are somewhat slower than the processor; thus, the latter waits for each analogue conversion or USART data transmission, and this results in a waste of time in some occasions. The use of interrupt-driven mode, nevertheless, would solve this problem.

The reason for choosing the single ADC mode is that this was the most suitable one for this application.

As mentioned before, the microcontroller being used is not fully exploited, and taking into consideration the cost increment between the previous circuit and this one, it would be a lucrative idea to take more advantage of it, using it in order to perform other kind of tasks - in this case the interrupts would be useful in order not to miss 'real time' response.

In a first draft of the code, the MPCM (Multiprocessor communication mode) by Atmega16 was used, which involves a 9th bits data frame. Finally it was removed because a microcontroller refers just to the PPC and not to other microcontrollers whereas it would be better if all devices in a network could communicate to each other.

Finally, comment that issues as errors and noise have not been investigated. Therefore, this can be an improvement in a future work as well.

6. Summary and conclusions

An electronic circuit is designed to be implemented in a soccer playing robot.

The main objective of this work is to make the robot know whether or not it is above a white line in the soccer field. In the bottom platform of the robot, there are 3 identical boards which have 8 infrared sensors and an Atmega16 microcontroller each. When the sensors are above a white line, they send a signal because the infrared signal will be reflected. On the other hand, weaker signals will be received when the sensors are on a green colour field. The knowledge of those sensors' values or states will help the robot to specify where it is located.

As the cards have to send the sensors' states digitally to the Power PC while they are read in an analogue way, it is necessary to calculate each sensor's threshold which varies according to the field the robot is playing on.

Each board will have its own address. The PPC will be sending orders to the network. Not only will the frame contain the address of the microcontroller which the order is for, but also the task that the Atmega16 has to perform; calculate the sensors' threshold or send data to the PPC.

The threshold calculation will take about two minutes; therefore the robot must be running around the field and taking samples from the sensors.

The information is sent to the PPC using the USART serial communication, with a start bit, 8 data, non-parity and one stop bit. The transmission is 9600 bps and the microcontrollers' connexion to the RS-485 network is been using a MAX489 transceiver.

All in all, this report explains how this work has been developed. The schematic is generated using Eagle software. Then the programme to be implemented in the microcontrollers is written in C language using the Win AVR compiler. The simulation part and the tests have been executed using AVR Studio, STK500 and Hyper Terminal.

In the tests, the PC has replaced the Power PC's role and just a point-to-point communication has been confirmed using RS-232. The result is that the microcontroller sends correctly the sensors' states after an adequate calibration.

The three boards are not yet built, but since the tests are so far successful, the entire network is very likely to work.

7. References

Literature

- [1] Alldatasheet.co.kr website. "OPB608 - Reflective Object Sensor - OPTEK Technologies". Feb. 2007
http://www.alldatasheet.co.kr/datasheet-pdf/pdf_kor/OPTEK/OPB607A.html
- [2] Atmel Corporation. "ATmega16(L)". Feb. 2007.
http://www.atmel.com/dyn/products/product_card.asp?part_id=2010
- [3] Atmel Corporation. "AVR JTAG ICE User Guide". Sep. 2001. April 2007.
http://www.atmel.com/dyn/products/tools_card.asp?family_id=607&family_name=AVR+8%2DBit+RISC+&tool_id=2737
- [4] Atmel Corporation, "STK500 User Guide". May 2007.
http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf
- [5] Axelson, Jan. Designing RS-485 Circuits. Circuit Cellar INK, the Computer Applications Journal. Issue 107 June 1999. Mar. 2007.
http://www.embeddedsys.com/subpages/resources/images/documents/microsys_art_RS485.pdf
- [6] Barnett, Richard, Larry O'Cull and Sarah Cox. Embedded C Programming and the Atmel Avr. 2003. Delmar Learning, Thomson, Canada.
- [7] Bies, Lammert. "RS485 serial information". Feb. 2007.
<http://www.lammertbies.nl/comm/info/RS-485.html>
- [8] B&B Electronics Mfg. "RS422/485 Application Note". Jun. 2006. Feb. 2007.
http://www.bb-elec.com/tech_articles/rs422_485_app_note/table_of_contents.asp
- [9] CadSoft Computer, Inc. "EAGLE 4.0 para Linux y Windows". 2000. April 2007.
<ftp://ftp.cadsoft.de/eagle/userfiles/doc/tutorial-spa.pdf>
- [10] Holmes, Steve. "C programming". University of Strathclyde Computer Centre. Jan. 1995. Feb. 2007.
<http://www.imada.sdu.dk/~svalle/courses/dm14-2005/mirror/c/>
- [11] Java Source and Support "C Tutorial" Feb. 2007.
<http://www.java2s.com/Tutorial/C/CatalogC.htm>
- [12] MAXIM - Dallas Semiconductor. "MAX489 Datasheet". Jun. 2003. Feb 2007.
http://www.datasheetcatalog.com/datasheets_pdf/M/A/X/4/MAX489.shtml

[13] Maxim Integrated Products, “Guidelines for Proper Wiring of an RS-485 (TIA/EIA-485-A) Network”. 12 July 2001. Feb 2007.
http://www.maxim-ic.com/appnotes.cfm?appnote_number=763&CMP=WP-1

[14] O'Flynn, Colin.” Downloading, Installing and Configuring WinAVR” 17th Mar. 2004. Feb. 2007.
http://winavr.sourceforge.net/install_config_WinAVR.pdf

[15] Wikipedia. “EIA-485”. Mar.2007
<http://en.wikipedia.org/wiki/RS-485>

Images

Figure 2.1 is from [1], p3
Figure 2.2 is from [1], p3
Figure 2.3 is from [1], p3
Figure 2.4 is from [2], p149
Figure 2.5 is from [2], p148
Figure 2.6 is from [2], p209
Figure 2.7 is from [2], p210
Figure 2.8 is from [8], Chapter 1, overview, p1
Figure 2.9 is from [15], p1
Figure 2.10 is from [7], p1
Figure 2.11 is from [13], p1
Figure 4.5a is from [2], p2
Figure 4.5b is from [1], p8
Figure 4.7 is from [3], p10

8. Appendix

Appendix A. C code

```
/**
 * Author: Miren Itsaso Martinez & David Calvo
 * Last update: 10th May 2007
 */

#include <mega16.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/wdt.h>
#include <util/delay.h>

#define EEMEN __attribute__((section(".eeprom")))

#define ENABLE 2
#define EN_TRANSISTOR 3

//Function declaration

void init_function(void);
unsigned int read_data(unsigned int);
void calibrar(void);
void result_calibration_eeprom(void);
void send_sensors_state(void);

static unsigned int val[8] __attribute__((section(".eeprom")));
unsigned char my_address, address, function, add_one, dares=0x00, my_UDR;
unsigned int data_sensor[8], result_c[8], i;

void init_function(void)
{
    WDTCSR=0x0F;
    my_address=0x01;

    for(i=0; i<=7; i++)
    {
        result_c[i]= eeprom_read_word((unsigned int*) &val[i]);
        wdt_reset ();
    }

    //Port Initialization
    PORTD=0x00;
    DDRD=0x0C; //PORTD.2 output(enable) and PORTD.3 also activates
    external transistor

    //Initialization UART
    UCSRA|=0x00;
    UCSRB|=(1<<RXEN)|(1<<TXEN);
    UCSRC|=(1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);
    UBRRH|=0x00;
    UBRRL=0x67; //16MHz, 9600baud

    //Initialization ADC but no START yet
    ADMUX=0x00;
    ADCSRA=0x87; //division factor 128./125kHz
}
```

```

// warm up the ADC, discard the first conversion
PORTB|=(1<<EN_TRANSISTOR);
ADCSRA |= (1 << ADSC);
do{}
while (ADCSRA & (1 << ADSC));
PORTB&=~(1<<EN_TRANSISTOR);
//Disable transmission
PORTB&=~(1<<ENABLE);
}

void calibrar(void)
{
    unsigned int h, leer_c[8], xprima[8], cont_field[8]={0,0,0,
0,0,0,0,0}, sum_field[8]={0,0,0,0,0,0,0,0}, average_field[8];
    unsigned int cont_line[8]={0,0,0,0,0,0,0,0}, sum_line[8]={0,0,0,
0,0,0,0,0}, average_line[8];
    unsigned int max[8]={0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000}, min[8]={0x03FF, 0x03FF, 0x03FF, 0x03FF, 0x03FF
, 0x03FF, 0x03FF, 0x03FF};

    for(h=0;h<=1000;h++)
    {
        wdt_reset ();
        for(i=0;i<=7;i++)
        {
            leer_c[i]=read_data(i);
            if(leer_c[i]>max[i])
            {
                max[i]=leer_c[i];
            }
            if(leer_c[i]<min[i])
            {
                min[i]=leer_c[i];
            }
            xprima[i]=((max[i]+min[i])>>1);
        }
        _delay_ms(50);
    }

    for(h=0;h<=1000;h++)
    {
        wdt_reset ();
        for(i=0;i<=7;i++)
        {
            leer_c[i]=read_data(i);
            if(leer_c[i]>=xprima[i])
            {
                cont_line[i]=cont_line[i]+1;
                sum_line[i]=sum_line[i]+leer_c[i];
                average_line[i]=sum_line[i]/cont_line[i];
            }
            else
            {
                cont_field[i]=cont_field[i]+1;
                sum_field[i]=sum_field[i]+leer_c[i];
                average_field[i]=sum_field[i]/cont_field[i];
            }
            result_c[i]=((average_line[i]+average_field[i])>>1);
        }
        _delay_ms(50);
    }
}

void result_calibration_eeprom(void)
{
    for(i=0;i<=7;i++)
    {
        wdt_reset ();
    }
}

```

```

        eeprom_write_word((unsigned int*) &val[i],result_c[i]);
    }
}

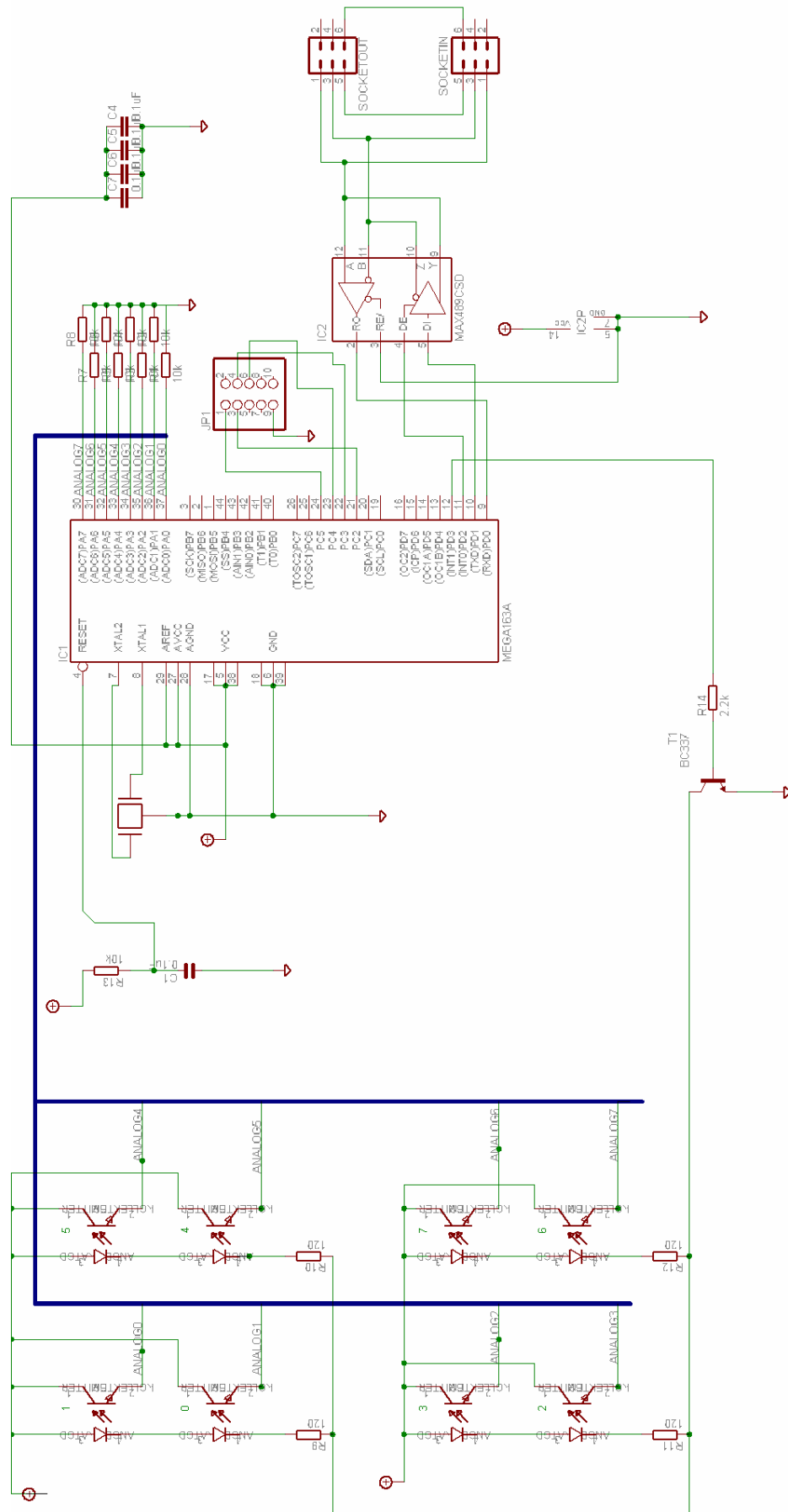
unsigned int read_data(unsigned int chanel)
{
    wdt_reset ();
    ADMUX=0x40;
    ADMUX|=(1<<REFS0) + chanel;
    PORTB|=(1<<EN_TRANSISTOR);
    _delay_ms(1);
    ADCSRA|=(1<<ADSC);
    do
    {}
    while(ADCSRA&(1<<ADSC));
    PORTB&=~(1<<EN_TRANSISTOR);
    return (unsigned int)ADCW;
}

void send_sensors_state(void)
{
    dades=0x00;
    for(i=0;i<=7;i++)
    {
        data_sensor[i]=read_data(i);
        if(data_sensor[i]>=result_c[i])
        {
            dades|=(1<<i);
        }
        else
        {
            dades=dades | 0b00000000;
        }
    }
    while ((UCSRA & (1 << UDRE)) == 0) {};// Do nothing until UDR is
    ready for more data to be written to it
    PORTB|=(1<<ENABLE);
    UDR=dades;
    while ((UCSRA & (1 << UDRE)) == 0) {}
    _delay_us(30);
    PORTB&=~(1<<ENABLE);
}

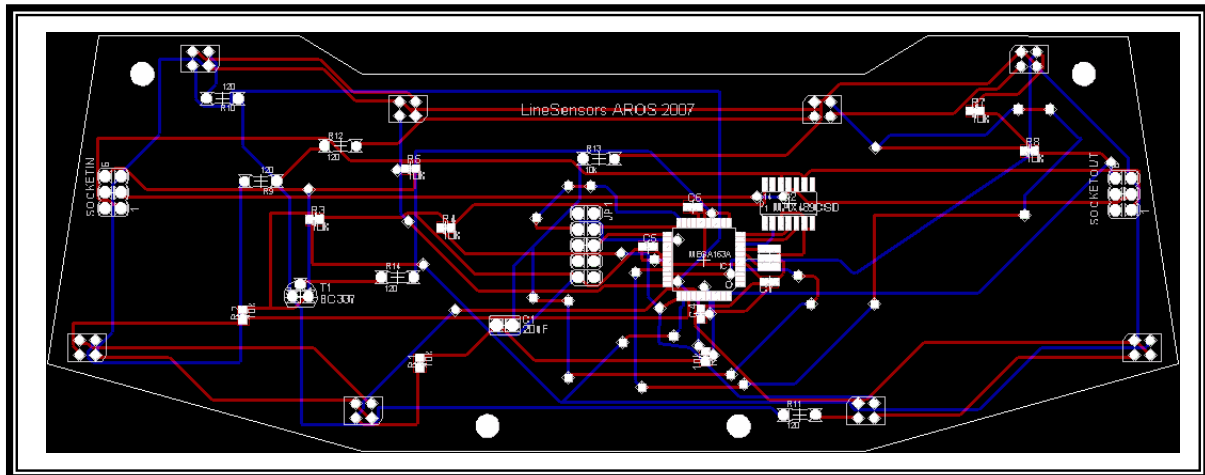
int main (void)
{
    init_function();
    while (1)
    {
        wdt_reset ();
        while ((UCSRA & (1 << RXC)) == 0) {};// Do nothing until
        data have been recieved and is ready to be read from UDR
        my_UDR=UDR;
        address=my_UDR&0X0f;
        function=my_UDR&0xf0;
        if((address==my_address)||(address==0x00))
        {
            if(function==0x80)
            {
                calibrar();
                result_calibration_eeprom();
            }
            else
            {
                send_sensors_state();
            }
        }
    }
}

```

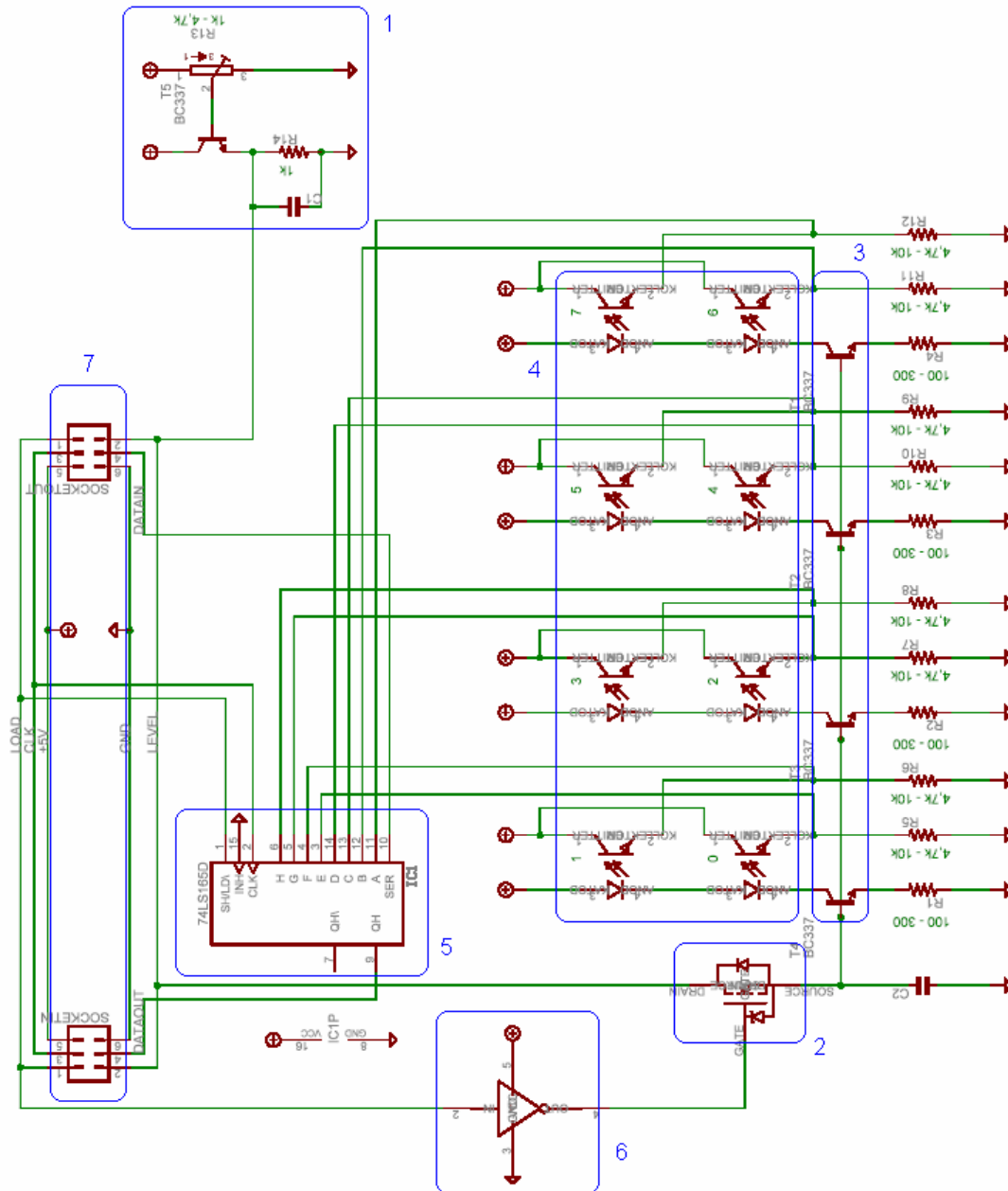
Appendix B. Schematic



Appendix C. Layout



Appendix D. Previous schematic



Appendix E. Required software and hardware

Eagle

EAGLE (Easily Applicable Graphical Layout Editor)
Version 4.01 for Windows
Professional Edition



- ★ This software is used for drawing the schematic circuit and making the layout.

Win AVR

Programmers Notepad 2 (AVR-GCC)

Version: v2.05.48

- ★ This program is used as a C compiler generating all file formats needed.

Mfile

- ★ Simple Makefile generator for AVR-GCC.



AVR Studio (Atmel)

AVRStudio4

Version 4.13



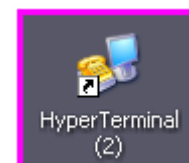
- ★ This software is used for:
 - o Simulation (it can not support all the microcontroller features).
 - o Programming the Atmega16:
 - In STK500 (SPI).
 - In the real board (JTAG).

Hyper terminal (Windows)

Hyper Terminal

Version 5.1

- ★ This software is used for communicating the PC with STK500 by UART/RS232.



STK500 (Atmel)

- ★ An evaluation kit used for testing the code.



JTAG ICE (Atmel)

- ★ Used for programming the microcontroller in the real board.



